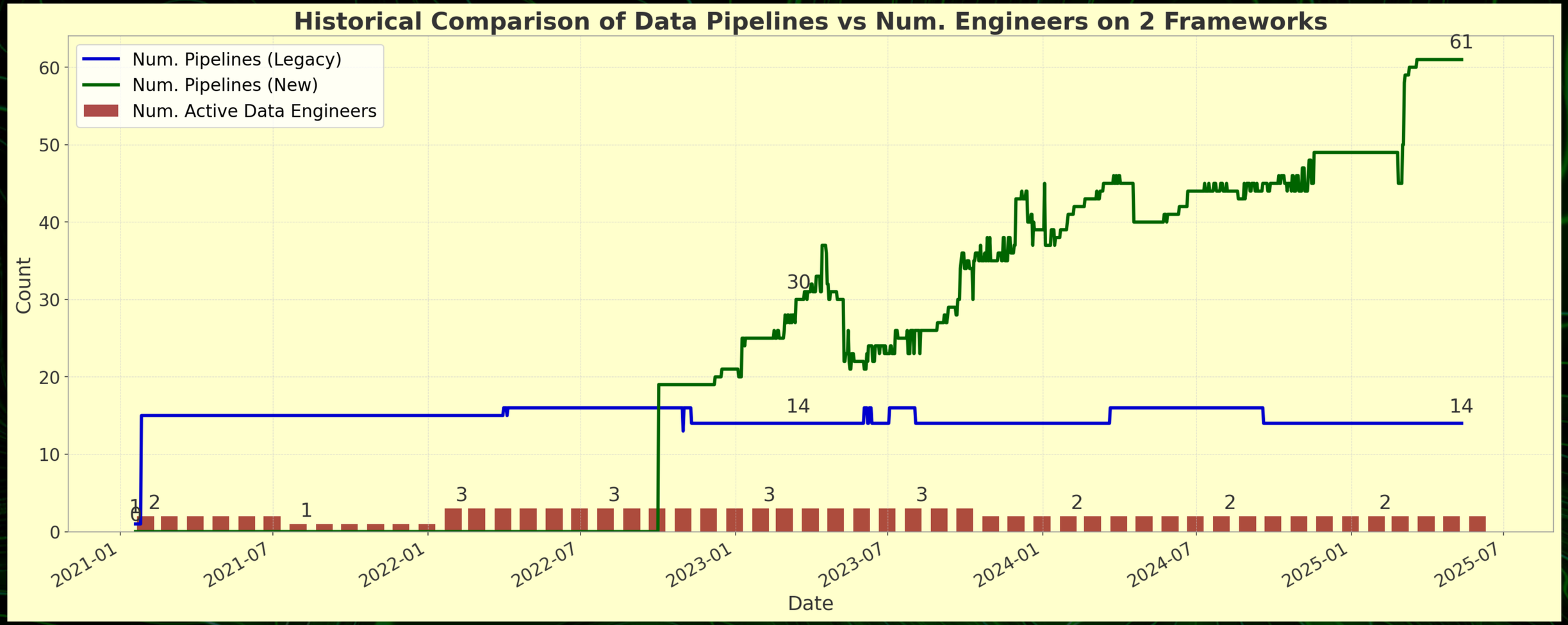
Bassoon: 低代码数据工程框架

Yueyang Chen

直接上效果展示

1. 数据工程师数量几乎不变，但数据管道持续增加
2. 记录在案的所有Global tech的系统故障、软件升级、数据迁移事件对增长曲线几乎没有影响



Bassoon如何提升工程效率 — 背后的理论

理论

事实：复杂的数据源 —> 数据工程 —> 无限可能的业务需求
推论：开发维护成本 = 数据源复杂性A x 业务需求复杂性B x (业务代码量C + 工程代码量D)
推论：在AB不变的前提下降低维护成本，唯一的办法是减低D

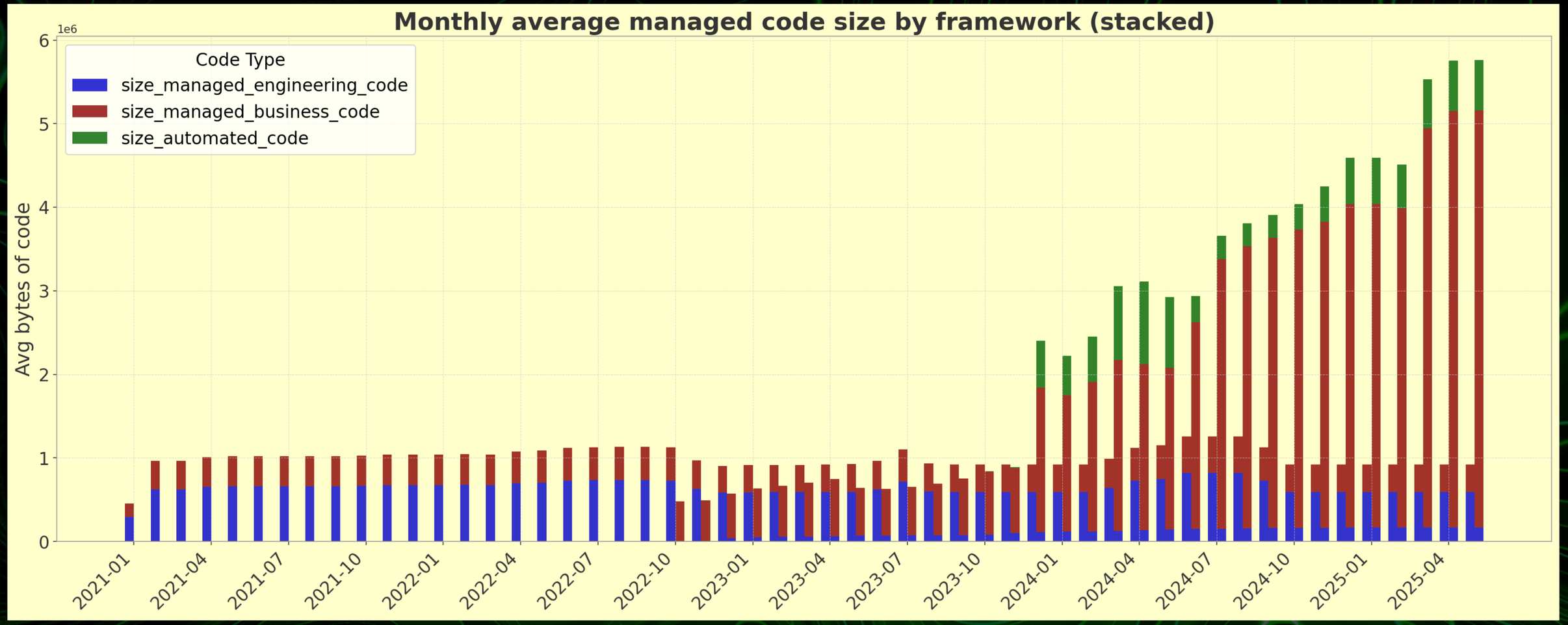
市面上各种框架的种种问题

1. 重度依赖于第三方平台（aws/azure/阿里云, linux/windows）-> 一旦老板想迁移平台，重写全部代码会让老板望而却步
2. 重度依赖于某种数据或业务 -> 一个框架也许只适用于某些公司的数据和业务，但不适用于其他公司
3. 工程代码和业务代码糅杂 -> 开发和维护人员都需要学习很多东西才知道怎么用，并且非技术人员参与难度大

核心解决方案：

1. 让**业务代码**成为数据工程师唯一需要考虑的东西
2. 可制定的业务规则不但包含数据如何流动如何清洗，还会包括用什么技术，在什么平台运行这些限制
3. 由bassoon来自动生成**工程代码**

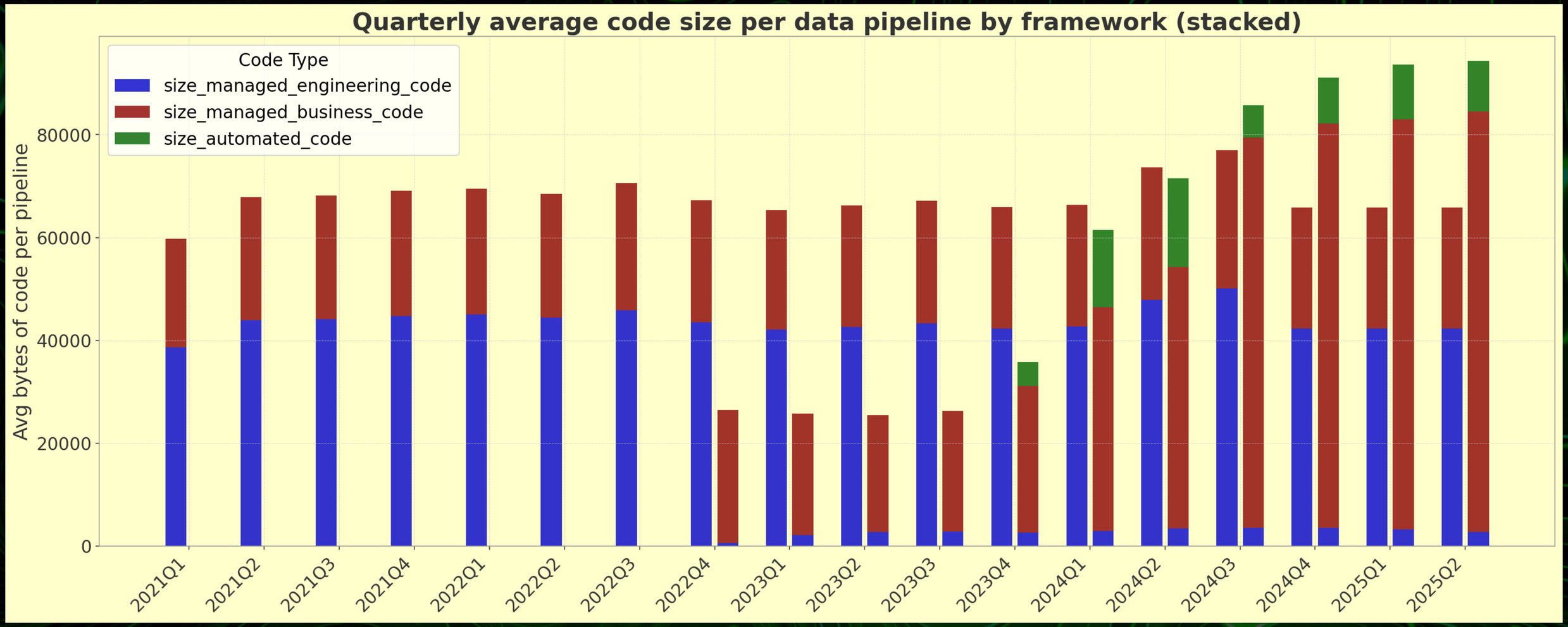
对比Bassoon和旧框架的代码总量分布



上图解说：

1. Pre-bassoon时代，蓝条比红条还要长，红条几乎不增长，说明维护工程代码耗费了工程师大量精力
2. Bassoon时代，蓝条被压缩到极致，红条不断增长，说明工程师精力更多花在了业务上，也说明更多人参与了业务开发

对比平均每个数据管道的代码量分布



上图解说：

工程代码量相比旧框架大幅降低，业务代码量大幅增加，说明我们用bassoon处理了比之前复杂很多的业务

Bassoon帮助我们在多次突发状况下保证业务稳定运作

- 2025-02-21, 一个AWS方面的故障导致某些老版的分布式机器用不了了，我们在**两个小时**内不但发现了故障来源，还将大量代码迁移到新版机器上继续运作，连给下游的预警信息都没发。
- 相比之下，AWS的人在**2025-03-04**才解决了那些老机器的问题。

- 2024年第一季度，Nike Global Tech要求所有团队升级一个公司要求使用的数据工具airflow，这次升级会影响我们当时所有的数据产品，并且原本预计会耗费大量的工程资源。但我们在**一个月**内完成了升级。
- 相比之下，我们的一个peer team（memberhub）在**次年8月份**才宣布完成了升级。

- 2024年第二季度，有一个下游团队强烈要求研究我们的某一些数据管道的代码。由于我们将所有代码集中管理，又不想将所有代码展示出去，所以这本来会要求我们将大段大段的代码挪到新的地方存储，然后重新部署重新测试，会耗费很多工程资源。
- 然而我们只花了**两天**就完成了迁移，和他们开始了长达数个月的深入到技术层面的交流合作。

我是谁，为什么开发了bassoon

- 经历过剧烈的技术动荡：在波士顿咨询（纽约）工作期间，我为多个行业的巨头公司服务过数字化转型。团队本来自研了一个ETL框架，但在每一个服务的客户期间，这个框架要不然因为重度依赖AWS，要不然因为客户公司的技术过于传统，都没能将框架用上，最后都是匆忙另做框架，惊险收尾。

- 见识过低工程化ETL的可怕：其中有两家客户将项目先交给Data Scientists进行项目的POC，然后将POC交给我们继续开发业务。独自完成了六千行pyspark的重构后，我深刻体会到了一个数据项目里工程人员和非工程人员的沟通其实有很大的gap。

- 入职耐克时我见到了一模一样的问题。本着对过往的恐惧，我2022年7月开始着手研发bassoon并在团队里推广使用。在后面数次Global Tech的软件升级和数据迁移中，我们用bassoon在工程资源有限、业务还要保证持续运作的前提下度过了一次次难关，证明了代码生成技术、工程业务分离 在技术动荡时期的强大韧性。